执行上下文中的变量对象详解

F7-罗纳尔多 2018-11-03 10:20:42 279 已收藏 2

文章标签： 前端 js 执行上下文 变量对象

版权

一个执行上下文的生命周期分为两个阶段

创建阶段

在这个阶段中，执行上下文会分别创建变量对象，建立作用域，以及确定this的指向。

1

代码执行阶段

创建完成后，就会开始执行代码，这个时候，会完成变量赋值，函数引用，以及执行其它代码。

1

变量对象（Variable Object）

变量对象的创建，依次经历了以下几个过程：

建立arguments对象。检查当前上下文中的参数，建立该对象下的属性与属性值。

检查当前上下文的函数声明，也就是使用function关键字声明的函数。在变量对象中以函数名建立一个属性，属性值为指向该函数所在内存地址的引用。如果函数名的属性已经存在，则会被新的引用所覆盖。

检查当前上下文中的变量声明，每找到一个变量声明，就在变量对象中以变量名建立一个属性，属性值为undefined。如果该变量名的属性已经存在（有个函数同名），为了防止同名的函数被修改undefined，则会直接跳过，原属性值不会被修改。

function foo() {console.log('function foo')};

var foo = 20; // 此为执行阶段 创建阶段跳过赋值

console.log(foo); // 20

1

2

3

foo的输出结果被覆盖是因为上面三条规则是适用于变量对象的创建过程中，也就是执行上下文的创建过程，而foo = 20 是在执行上下文的过程中运行的，输出结果就是20。对比下面的例子：

console.log(fooAn); // function fooAn() {console.log('function fooAn')}

function fooAn() {

console.log('function fooAn')

}

var fooAn = 30;

1

2

3

4

5

// 上面例子的执行顺序

// 首先将所有函数声明放入变量对象中

function fooAn() {

console.log('function fooAn')

}

// 其次将所有的变量声明放入变量对象中，但是因为fooAn有同名函数，所以此时会跳过undefined的赋值

// 继续执行代码

console.log(fooAn); // function fooAn() {console.log('function fooAn')}

fooAn = 30;

1

2

3

4

5

6

7

8

9

10

11

12

所以用图表示：

在上面的例子中可以看到，function声明会比var声明优先级更高一点。

// demo1

function test1() {

console.log(a); // undefined

console.log(foo1()); // 2

var a = 1;

function foo1() {

return 2;

}

}

test1();

1

2

3

4

5

6

7

8

9

10

11

在上例中，我们直接从test1（）执行的上下文开始。全局作用域中运行test1（）时，test1（）的执行上下文开始创建：

// 创建过程

test1EC = {

// 变量对象

VO: {},

scopeChain: {}

}

// VO 为Variable Object的缩写，即变量对象

VO = {

arguments: {...}, // 在浏览器的展示中，函数的参数可能并不是放在arguments对象中，这里为了方便理解，我做了这样的处理

foo1: <foo1 reference>, // 表示foo1地址的引用

a: undefined

}

1

2

3

4

5

6

7

8

9

10

11

12

13

未进入执行阶段之前，变量对象中的属性都不能访问！但是在进入执行阶段后，变量对象转变为了活动对象，里面的属性都能被访问了，然后开始进行执行阶段的操作。

变量对象和活动对象其实都是同一个对象，只是处于执行上下文的不同生命周期。不过只有处于函数调用栈栈顶的执行上下文中的变量对象，才会变成活动对象。

// 执行阶段

VO -> AO // Active Object

AO = {

arguments: {...},

foo1: <foo1 reference>,

a: 1,

this: window

}

1

2

3

4

5

6

7

8

因此上面的例子执行顺序就变成了这样：

function test1() {

function foo1() {

return 2;

}

var a;

console.log(a);

console.log(foo1());

a = 1;

}

test1();

1

2

3

4

5

6

7

8

9

10

11

全局上下文的变量对象

以浏览器中为例，全局对象为window。

全局上下文有一个特殊的地方，他的变量对象，就是window对象。而这个特殊，在this的指向上也同样适用，this的指向也是window。

// 以浏览器中为例，全局对象为window

// 全局上下文

windowEC = {

VO: Window,

scopeChain: {},

this: Window

}

1

2

3

4

5

6

7

除此之外，全局上下文的生命周期与程序的生命周期一致，只要程序运行不结束，比如关掉浏览器窗口，全局上下文就会一直存在。其它所有的上下文环境都能直接访问全局上下文的属性。

————————————————

版权声明：本文为CSDN博主「F7-罗纳尔多」的原创文章，遵循CC 4.0 BY-SA版权协议，转载请附上原文出处链接及本声明。

原文链接：https://blog.csdn.net/wuyifei724277441/article/details/83684657